QUESTION 1

No relational DBMS

- MongoDB, Couchbase, CouchDB

Data Model: Store data in semi-structured documents, typically in JSON or BSON format. Documents can have different structures within the same collection.

Use Cases: Content management systems, catalogs, real-time analytics, and applications with evolving schemas. And this fall in the

- Amazon DynamoDB, Riak
  Data Model: Data is stored as key-value pairs, where each key is associated with a value. The value can be a simple scalar or complex data structure.
  Use Cases: Caching, session management, real-time analytics, and distributed data stores.

- Apache Cassandra, HBase, ScyllaDB
  Data Model: Data is organized into column families, which contain rows with columns. Each row can have a different set of columns, making it flexible for different data types.
  Use Cases: Time-series data, sensor data, event logging, and applications requiring high write throughput.

- Neo4j, Amazon Neptune, OrientDB
  Data Model: Data is represented as nodes, edges, and properties. Graph databases excel at storing and querying relationships between data entities.
  Use Cases: Social networks, recommendation engines, fraud detection, and knowledge graphs.

- InfluxDB, OpenTSDB, TimescaleDB
  Data Model: Optimized for storing time-series data, such as sensor readings, metrics, and logs. Data is typically indexed by time.
  Use Cases: IoT applications, monitoring and observability, and analyzing time-series data.

- db4o (discontinued), Versant
  Data Model: These databases store data as objects with attributes and methods. They are used less frequently compared to other NoSQL types.
  XML and JSON Databases.

- BaseX (XML), ArangoDB (JSON)
  Data Model: Designed to store and query XML or JSON data, respectively. They provide efficient storage and retrieval of hierarchical data.
  NewSQL Databases.

- Google Spanner, CockroachDB
  Data Model: These databases aim to combine the benefits of traditional relational databases with horizontal scalability and global distribution.

NoSQL databases are often chosen based on the specific requirements of an application, such as data structure, scalability, and performance. Organizations may also use multiple NoSQL databases or a combination of NoSQL and relational databases within their technology stack to meet different data management needs. The choice of NoSQL database depends on factors like data volume, data complexity, query patterns, and scalability requirements.

And for the best if your data is primarily document-based, a document-oriented database like MongoDB

QUESTION 2

In Java, when you execute an SQL statement using Statement or PreparedStatement, the method execute() or executeUpdate() returns a boolean value to indicate whether the execution was successful. The return value is true if the execution resulted in a ResultSet (for queries) or false if it's an update statement (INSERT, UPDATE, DELETE) or if there's an error.

Here's how it works:

true when a ResultSet is generated:

For Statement and PreparedStatement objects used for SELECT queries, execute() can return true because a ResultSet is generated when the query is executed successfully. This means there are rows of data to be fetched from the database.
false when there's no ResultSet:

For Statement and PreparedStatement objects used for INSERT, UPDATE, DELETE, or other non-query operations, executeUpdate() or execute() returns false because these statements do not produce a ResultSet. Instead, they return the number of rows affected by the operation.
false when there's an error:

If there is an error during the execution of the SQL statement (e.g., syntax error, database connection issue, etc.), execute() or executeUpdate() will return false to indicate that the operation failed.
So, the return value of execute() or executeUpdate() depends on the type of SQL statement and whether the execution was successful or not. If you are observing unexpected behavior, make sure to check for error messages or exceptions to identify the specific issue causing the unexpected result.

The execute() method from a PreparedStatement or Statement in Java should not return false when the condition is true for executing a valid SQL statement.

If execute() returns false, it typically indicates that there was an issue with the execution of the SQL statement.

QUESTION 3

In Java programming, "arguments" and "parameters" refer to different aspects of passing data into methods or functions. Here are their definitions and the difference between them:

Parameters:

Parameters are variables declared in the method signature or function declaration.
They act as placeholders for values that will be provided when the method or function is called.
Parameters are defined with their data types and names.
Parameters are part of the method's or function's definition and are used to specify what kind of data the method or function expects.
Example of method declaration with parameters:

```
java
Copy code
public void printMessage(String message) {
   // ...
}
```
Arguments:

Arguments are the actual values or expressions passed to a method or function when it is called.
They provide the concrete data that will be used inside the method or function.
Arguments are supplied when invoking the method or function, and they must match the order, number, and data types of the parameters declared in the method or function definition.
Example of calling a method with arguments:

```
java
Copy code
printMessage("Hello, World!"); // "Hello, World!" is the argument
```
In summary, parameters are the placeholders defined in the method or function declaration, while arguments are the actual values or expressions that are provided when the method or function is called. Parameters specify what kind of data is expected, and arguments provide the specific data to be used within the method or function. The two terms are often used in conjunction when discussing method invocation in Java.

QUESTION 4

In Java programming, you can validate user input to ensure that the provided input for an ID does not exceed a certain length (e.g., maximum character limit) using a variety of approaches. Here's an explanation of how you can perform this validation:

Using String Length Check:
You can check the length of the provided ID as a string and compare it to the maximum allowed length. For example:

java
Copy code
```
public boolean isIdValid(String id, int maxLength) {
    return id.length() <= maxLength;
}
```
In this example, the isIdValid method takes the ID and a maximum length as arguments and returns true if the ID length is less than or equal to the specified maximum length.

Using Regular Expressions:
You can also use regular expressions to validate the format and length of the ID. For example, to ensure that the ID consists of only digits and has a maximum length of 10 characters:

java
Copy code
```
public boolean isIdValid(String id) {
    return id.matches("\\d{1,10}");
}
```
This regular expression (\\d{1,10}) checks that the ID contains between 1 and 10 digits.

Handling Validation in User Input Forms:
If you're building a user interface (UI) with input fields, you can set the maximum input length directly in the UI components. For example, in HTML, you can use the maxlength attribute:

html
Copy code
```
<input type="text" id="userId" name="userId" maxlength="10">
```
This HTML input field restricts user input to a maximum of 10 characters.

Displaying Error Messages:
When performing validation, it's essential to provide meaningful error messages to inform users about validation failures. If the ID is too long, you can display an error message indicating that the ID should not exceed the maximum length.

Here's an example of how to use the isIdValid method to check if a provided ID is valid:

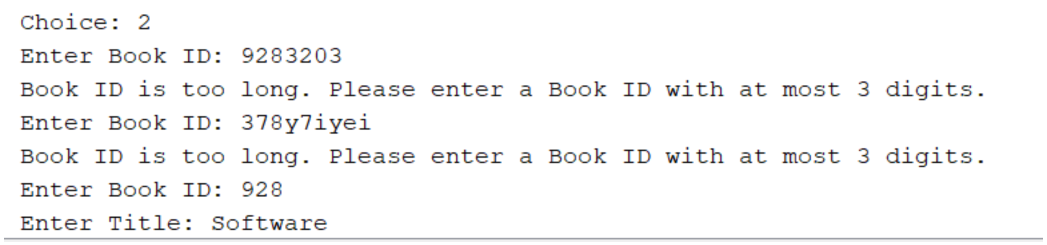
java
Copy code
```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
```

```
  System.out.print("Enter an ID: ");
  String userId = scanner.next();

  if (isIdValid(userId, 10)) {
    System.out.println("Valid ID.");
  } else {
    System.out.println("Invalid ID. It should not exceed 10 characters.");
  }
}
```
In this example, the program checks if the provided user ID is valid (not exceeding 10 characters) and displays an appropriate message based on the validation result.

<u>Here are the codes I used to validate my program</u>

```
try {
             while (true) {
             System.out.print("Enter Book ID: ");
             bookId = sc.next();

             if (bookId.length() != 3) {
               System.out.println("Book ID is too long. Please enter a Book ID with at most 3
digits.");
                 } else if (!bookId.matches("\\d+")) {
               System.out.println("Invalid input. Please enter a valid numeric Book ID.");
                  } else {
               System.out.print("Enter Title: ");
                     title = sc.next();

             Connection con = DriverManager.getConnection(dbUrl, username, passwd);
            Statement st = con.createStatement();

           String insertSql = "INSERT INTO book (book_id, title) VALUES ('" + bookId + "', '" +
title + "')";
            int rowAffected = st.executeUpdate(insertSql);

            if (rowAffected >= 1) {
               System.out.println("Data Saved!");
               } else {
                  System.out.println("Data not Saved!");
              }

               con.close();

               System.out.print("Enter Yes or No to continue or to quit: ");
            String answer = sc.next().toLowerCase();
        if (answer.equalsIgnoreCase("yes")) {
            System.out.println("Thank you for using the system");
```

<u>The screenshot of the results</u>



```
Choice: 2
Enter Book ID: 9283203
Book ID is too long. Please enter a Book ID with at most 3 digits.
Enter Book ID: 378y7iyei
Book ID is too long. Please enter a Book ID with at most 3 digits.
Enter Book ID: 928
Enter Title: Software
```